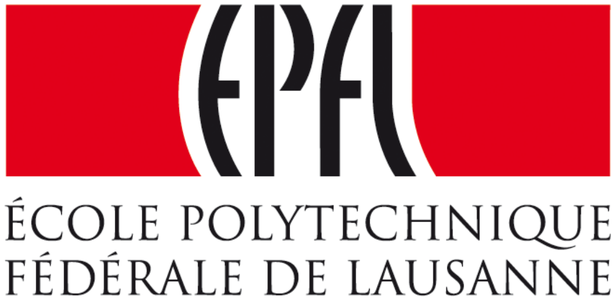
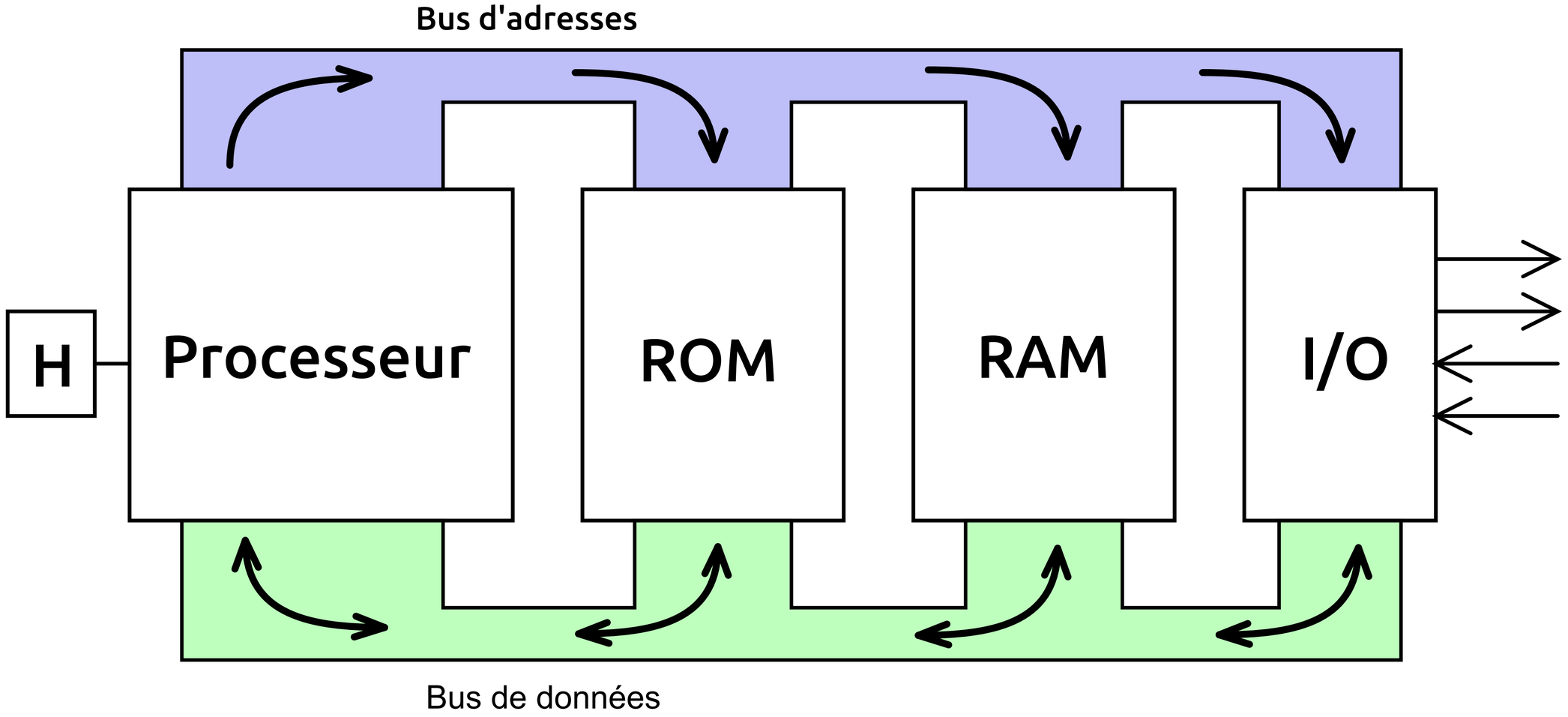


Microinformatique pour GM

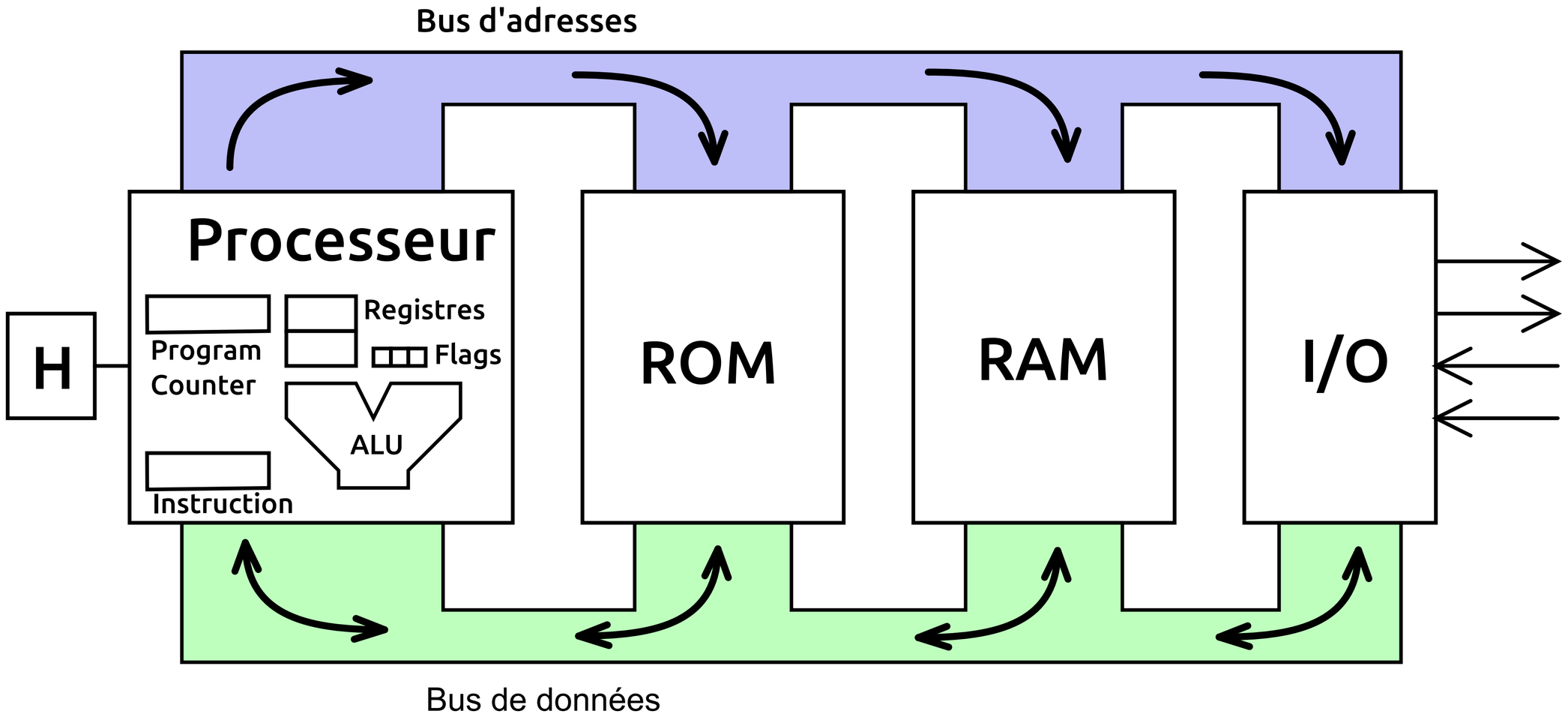
Découverte de l'assembleur

Pierre-Yves Rochat

Architecture d'un système informatique



Le processeur(CPU) exécute des instructions

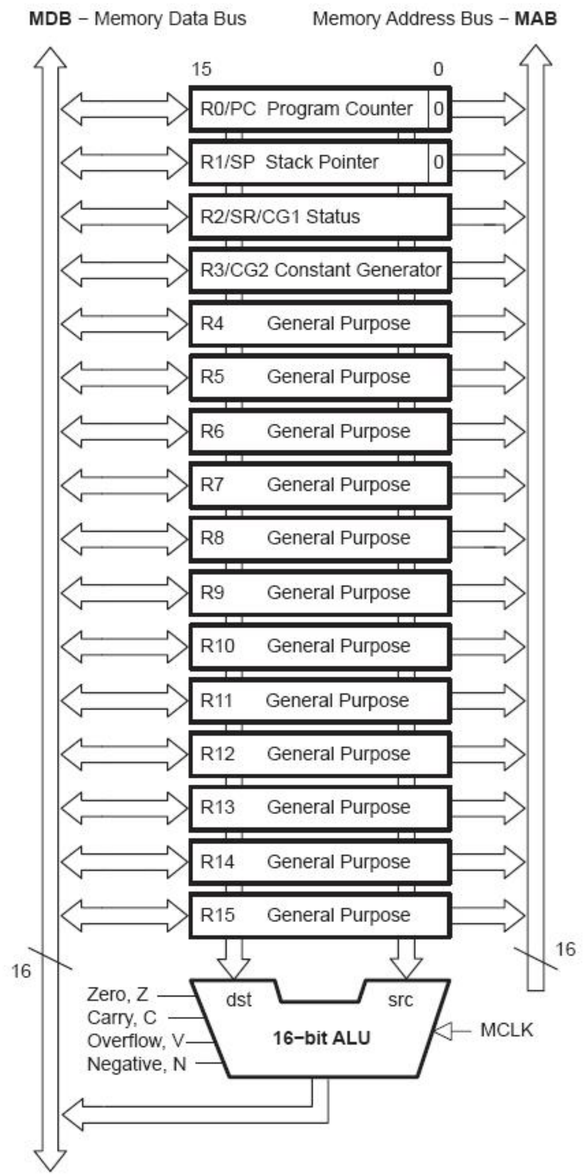


- Copies de données
- Opérations arithmétiques
- Opérations logiques
- Sauts

Choix des instructions

- La liberté du concepteur du processeur !
- Schémas logiques très complexes..
- Utilisation de la plage binaire
- Ex : 65'536 instructions (16 bits)
- ...27 instructions pour le MSP430 !

- Architecture du MSP430 :



Codage des instrucations

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MSP430 instruction set

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	**0**	**0**	**1**	**0**	**0**	**opcode**			**B/W**	**As**		**register**				**Single-operand arithmetic**
0	0	0	1	0	0	0	0	0	B/W	As		register				**RRC** Rotate right (1 bit) through carry
0	0	0	1	0	0	0	0	1	0	As		register				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		register				**RRA** Rotate right (1 bit) arithmetic
0	0	0	1	0	0	0	1	1	0	As		register				**SXT** Sign extend byte to word
0	0	0	1	0	0	1	0	0	B/W	As		register				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		register				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC
0	**0**	**1**	**condition**			**10-bit signed offset**										**Conditional jump; PC = PC + 2×offset**
0	0	1	0	0	0	10-bit signed offset										**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset										**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset										**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset										**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset										**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset										**JGE** Jump if greater or equal
0	0	1	1	1	0	10-bit signed offset										**JL** Jump if less
0	0	1	1	1	1	10-bit signed offset										**JMP** Jump (unconditionally)
opcode				**source**					**Ad**	**B/W**	**As**	**destination**				**Two-operand arithmetic**
0	1	0	0	source					Ad	B/W	As	destination				**MOV** Move source to destination
0	1	0	1	source					Ad	B/W	As	destination				**ADD** Add source to destination
0	1	1	0	source					Ad	B/W	As	destination				**ADDC** Add source and carry to destination
0	1	1	1	source					Ad	B/W	As	destination				**SUBC** Subtract source from destination (with carry)
1	0	0	0	source					Ad	B/W	As	destination				**SUB** Subtract source from destination
1	0	0	1	source					Ad	B/W	As	destination				**CMP** Compare (pretend to subtract) source from destination
1	0	1	0	source					Ad	B/W	As	destination				**DADD** Decimal add source to destination (with carry)
1	0	1	1	source					Ad	B/W	As	destination				**BIT** Test bits of source AND destination
1	1	0	0	source					Ad	B/W	As	destination				**BIC** Bit clear (dest &= ~src)
1	1	0	1	source					Ad	B/W	As	destination				**BIS** Bit set (logical OR)
1	1	1	0	source					Ad	B/W	As	destination				**XOR** Exclusive or source with destination
1	1	1	1	source					Ad	B/W	As	destination				**AND** Logical AND source with destination (dest &= src)

Codage des instrucations

opcode				source	Ad	B/W	As	destination	Two-operand arithmetic
0	1	0	0	source	Ad	B/W	As	destination	**MOV** Move source to destination
0	1	0	1	source	Ad	B/W	As	destination	**ADD** Add source to destination
0	1	1	0	source	Ad	B/W	As	destination	**ADDC** Add source and carry to destination
0	1	1	1	source	Ad	B/W	As	destination	**SUBC** Subtract source from destination (with carry)
1	0	0	0	source	Ad	B/W	As	destination	**SUB** Subtract source from destination
1	0	0	1	source	Ad	B/W	As	destination	**CMP** Compare (pretend to subtract) source from destination
1	0	1	0	source	Ad	B/W	As	destination	**DADD** Decimal add source to destination (with carry)
1	0	1	1	source	Ad	B/W	As	destination	**BIT** Test bits of source AND destination
1	1	0	0	source	Ad	B/W	As	destination	**BIC** Bit clear (dest &= ~src)
1	1	0	1	source	Ad	B/W	As	destination	**BIS** Bit set (logical OR)
1	1	1	0	source	Ad	B/W	As	destination	**XOR** Exclusive or source with destination
1	1	1	1	source	Ad	B/W	As	destination	**AND** Logical AND source with destination (dest &= src)

Codage des instrucations

0	0	1	condition			10-bit signed offset	Conditional jump; PC = PC + 2×offset
0	0	1	0	0	0	10-bit signed offset	**JNE/JNZ** Jump if not equal/zero
0	0	1	0	0	1	10-bit signed offset	**JEQ/JZ** Jump if equal/zero
0	0	1	0	1	0	10-bit signed offset	**JNC/JLO** Jump if no carry/lower
0	0	1	0	1	1	10-bit signed offset	**JC/JHS** Jump if carry/higher or same
0	0	1	1	0	0	10-bit signed offset	**JN** Jump if negative
0	0	1	1	0	1	10-bit signed offset	**JGE** Jump if greater or equal
0	0	1	1	1	0	10-bit signed offset	**JL** Jump if less
0	0	1	1	1	1	10-bit signed offset	**JMP** Jump (unconditionally)

Codage des instrucations

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instruction
0	**0**	**0**	**1**	**0**	**0**	**opcode**			**B/W**	**As**		**register**				**Single-operand arithmetic**
0	0	0	1	0	0	0	0	0	B/W	As		register				**RRC** Rotate right (1 bit) through carry
0	0	0	1	0	0	0	0	1	0	As		register				**SWPB** Swap bytes
0	0	0	1	0	0	0	1	0	B/W	As		register				**RRA** Rotate right (1 bit) arithmetic
0	0	0	1	0	0	0	1	1	0	As		register				**SXT** Sign extend byte to word
0	0	0	1	0	0	1	0	0	B/W	As		register				**PUSH** Push value onto stack
0	0	0	1	0	0	1	0	1	0	As		register				**CALL** Subroutine call; push PC and move source to PC
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	**RETI** Return from interrupt; pop SR then pop PC

Exemple de programme

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

```
;-------------------------------------------------------------------------------
; MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;-------------------------------------------------------------------------------
            .cdecls C,LIST,"msp430.h"       ; Include device header file

;-------------------------------------------------------------------------------
            .def    RESET                   ; Export program entry-point to
                                            ; make it known to linker.
;-------------------------------------------------------------------------------
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.
;-------------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer

;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            mov.b   #0x3F,&P1DIR                ; P1.0 à P1.5 en sortie
Boucle      inc.b   &P1OUT                      ; incrémente la valeur affichée sur les LED
            mov.w   #20000,R4
Att         dec.w   R4                          ; attente
            jne     Att
            jmp     Boucle
            nop

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
            .global __STACK_END
            .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
            .sect   ".reset"                ; MSP430 RESET Vector
            .short  RESET
```

Exemple de programme

```
1  ;-------------------------------------------------------------------
2  RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
3  StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer
4
5
6  ;-------------------------------------------------------------------
7  ; Main loop here
8  ;-------------------------------------------------------------------
9
10             mov.b   #0x3F,&P1DIR    ; P1.0 à P1.5 en sortie
11 Boucle      inc.b   &P1OUT          ; incrémente la valeur affichée sur les LED
12             mov.w   #20000,R4
13 Att         dec.w   R4              ; attente
14             jne Att
15             jmp Boucle
16             nop
```